
TURING
图灵教育
计算机程序设计竞赛权威指导书
新编
实用算法分析
与程序设计
王建德 吴永辉 编著
陈越 审校
人民邮电出版社
POSTS & TELECOM PRESS

新编
实用算法分析与程序设计

作者编著的《实用算法的分析与程序设计》一书曾经在全国信息学奥林匹克竞赛产生了广泛和深远的影响。本书是作者在该书基础上十年磨一剑、精心编写而成的，反映了近年来程序设计教育和竞赛培训活动出现的新趋势。全书不仅从教学的角度详细讲解算法的理论，而且从竞赛的角度对经典习题进行详细解析，重在培养学生灵活运用算法的能力。

本书是一部优秀的算法参考书，更是各层次程序设计竞赛培训不可错过的辅导书。

本书特色

- 采用结构清晰、移植性强且贴近自然语言表述的类程序设计语言。
- 各章节之间有着紧密的内在联系，但是彼此又相对独立。
- 例题多采用一题多解、多向求解的方法，且各章均有与其内容相匹配的练习题。
- 开辟专门网站（http://admis.fudan.edu.cn/publications.htm），为读者提供大量的经典例题和测试数据。

王建德 著名的信息学奥林匹克竞赛金牌教练，国务院特殊津贴专家，中学特级教师。他所辅导的学生在国际奥林匹克信息学竞赛（IOI）中获7金、2银、2铜的优异成绩。先后出版了22本关于程序设计和算法的学术专著，其中《实用算法的分析与程序设计》广受好评，长期以来是国内各类程序设计竞赛的必备教程。

吴永辉 博士，复旦大学计算机科学与工程系副教授，ACM-ICPC中国赛区指导委员会（ACM-ICPC Council China）成员，复旦大学ACM程序设计竞赛队教练。自2001年起连续带队进入ACM-ICPC世界总决赛，并取得过世界第6名的佳绩。主要研究方向为数据库，在《计算机研究与发展》、《软件学报》以及重大学术会议上发表多篇论文，参与译著《数据通信与网络》和《数据通信、计算机网络与开放系统》。

本书相关信息请访问：图灵网站 http://www.turingbook.com
读者/作者热线：(010)88593802
反馈/投稿/推荐信箱：contact@turingbook.com

上架建议 计算机/计算机科学/计算机算法

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-17706-3

9 787115 177063 >

ISBN 978-7-115-17706-3/TP

定价：39.00 元

TURING
图灵教育

新编
实用算法分析与程序设计

王建德 吴永辉 编著
陈越 审校

人民邮电出版社
北 京

图书在版编目（CIP）数据

新编实用算法分析与程序设计 / 王建德，吴永辉编著．
—北京：人民邮电出版社，2008.7
ISBN 978-7-115-17706-3

Ⅰ.新… Ⅱ.①王…②吴… Ⅲ.①电子计算机－算法分析－高等学校－教材 ②电子计算机－程序设计－高等学校－教材 Ⅳ.TP301.6 TP311

中国版本图书馆 CIP 数据核字（2008）第 024891 号

内 容 提 要

本书是一部程序设计竞赛教程。书中首先讲述了算法的基本概念、各种排序与解题的方法及策略，然后论述了初等数论、计算几何学、搜索和图论的有关算法，最后讨论了动态规划。本书不仅从教学的角度详细讲解算法理论，而且从竞赛的角度对经典习题进行详细解析，培养学生灵活运用算法的能力。

本书既可以作为大专院校计算机专业算法类课程的教材，亦可以作为大中学校计算机竞赛活动的培训教材，还可供计算机软硬件研发人员参考。

新编实用算法分析与程序设计

◆ 编　　著　王建德　吴永辉
审　　校　陈　越
责任编辑　杨海玲　刘　静

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷
新华书店总店北京发行所经销

◆ 开本：800×1000　1/16
印张：21
字数：515千字　2008 年 7 月第 1 版
印数：1－4000册　2008 年 7 月河北第 1 次印刷

ISBN 978-7-115-17706-3 /TP

定价：39.00元

读者服务热线：(010)88593802　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

十年前，我们为全国奥林匹克信息学活动编著了一本教材《实用算法的分析与程序设计》，虽然，该书早已在市场上售罄，但是它在编程解题能力的形成方面对许多读者产生了深远的影响。这些影响并没有随着岁月的流逝而淡漠。如果你上网“google”相关信息，屏幕上就会出现许多对该书赞美和留恋的贴子，而且各种形式的同名电子图书不计其数。确实，该书涉及的算法有极大的实用价值，在当时历史条件下属于相当新颖和先进的。这正是受读者青睐的一个重要原因。近几年来，不少出版社主动找上门来，希望我们能够顺应读者要求修订再版该书。这件事一直拖延至今，缘由是在如何修订再版的问题上，我们不想做简单的重复或细枝末节上的修改，而是想对该书来一个“更新换代”，力图用更好的方式回馈读者的期盼。十年来，随着算法讨论的深入和各类竞赛活动的推波助澜，涌现出许多新的算法思想。例如，在繁杂的数据中寻找和利用有价值的序，使问题获得简化；对无序的运算对象进行二分搜索；在满足单调性条件的最优性问题上使用参数搜索；通过Dinic算法分阶段、分层次地改进网络流图中的流量，计算最大流；利用四边形不等式原理推出最优决策的单调性，从而改善动态规划的效率，等等。与算法相关的经典例题也层出不穷。《实用算法的分析与程序设计》的新版本应该与时俱进，将这些新知识、新成果吸纳进来，并且以深入浅出、贴近实际的形式展现给读者。

为了使选用本书的教师容易教，学生容易学，软硬件研发人员查阅时容易懂，本书在表述上采取了如下措施。

(1) 全书以某一方面的算法为基本构件，各章节既互相联系又相对独立，便于教师按照不同培养目标组织教学，方便读者根据需要选择学习内容。

(2) 对每个算法的原理进行必要的分析和证明，定理证明大多采用初等数学的分析方法，公式推导尽可能做到浅显和详细。通过“程序伪代码”说明其工作过程，并给出了运行时间的详细分析。对其中一些复杂算法附加了图示，使其过程更加具体、直观和形象。每个算法都有具体的应用例证来帮助理解。

(3) 对于一些例题，我们尽量采用“一题多解”或“多向求解”，并且尽量结合算法分析讲解一些常用的思维方式和解题策略，拓宽读者的思路，使其学会如何应用算法知识解题，如何选择有效算法。每章末尾提供一些习题，这些习题与该章讲授的内容顺序相匹配。为了使读者有更多的实践机会，我们开辟了一个专门的网站http://admis.fudan.edu.cn/publications.htm，为读者提供了大量的经典例题和测试数据。

(4) 为了使程序样例不受单一语言的局限，不让某种特定语言的特殊性掩盖算法的本质内容，本书中采用了一种结构清晰、移植性强且贴近自然语言表述的类程序设计语言。只要你具备Pascal、C、true.bascal等任一种语言的基础，就可以比较轻松地读懂其语义。本书之所以未给出

可直接编译运行的程序代码，除了考虑到语言的通用性外，还有一个重要的原因，就是想给读者留出动手编程的空间，避免无意义的“依葫芦画瓢”。

根据知识难度和“因材施教”的原则，本书既可以作为大专院校计算机专业算法类课程的教材，亦可作为大中学校计算机竞赛活动的培训教材，还可供计算机软硬件研发人员参考。建议教师组织教学或读者自学本书时，注意以下3个方面的问题。

(1) 培养良好的认知结构。初学时不必强求记住所有名词，在遇到难以理解的概念时也不必强求立即学会。可以和周围的人讨论，也可以先做一个标记接着往下读，在学习一个阶段后再回过头来思考当初的问题，这样可能会“茅塞顿开”。实际上，书中介绍的每一个算法都值得初学者多读几遍，这样既有助于记忆又可以获得更深入的理解。在深入学习算法的过程中，要注重相关算法理论的研究，通性通法，并通过不断地应用得以强化，使得相关理论和通性通法形成一个纵横交错、脉络分明、结构合理的知识网络。同时注意把一些常用的解题技巧和变换方法放在记忆库里，把同类问题和同类方法“存储在一起”，使知识条理化和系统化。

(2) 按照“双轨复式”的方式学习算法。所谓“双轨”是指学习内容上沿两条线：一是知识结构，二是经典试题。在学习每一个知识点的同时，积累相关经典试题的解析经验，进行系统的、集中的分类解题训练。解一道试题，从某些侧面窥悉相关算法的基本特征。解同类算法的一组试题，通晓该算法知识的整体结构以及与其他知识点之间的联系。所谓“复式”是指某些内容特别重要的或较复杂的算法知识，不能指望经过一两次做题就能融汇贯通、熟练掌握，必须根据量力而行的原则，针对自己的认知水平和智力发展水平把同一内容分成深度与广度不同的若干片断，一步步地螺旋式上升，最后达到较好掌握的目的。注意把握节奏，一个阶段一个中心，问题过难过易都不能出现在一段较长的时间里，过难会丧失信心，过易则难以激发热情，这就需要读者从读书和做题过程中积极体验，认真反思，获取反馈，合理调整。

(3) 在学习算法理论的同时要多解题，从书本知识和编程实践中寻找再发现与再创造的契机。既要注意运算结果的正确性，也要注意知识产生和应用的过程性（概念和法则被概括的过程、数据关系被抽象的过程，以及解题思路逐步展开的过程）。要细化书本知识，如同电视屏幕中体育大赛的慢镜头式的分解，真正使自己学有所得，学有所悟。要多角度地思考问题。同一个问题，可选择的算法可能不止一种。不管所要完成的任务是大是小，最好在完成之后再设法寻找另一种方法完成它，后一种方法可能是更好的方法。特别是在使用某方法却没有成功时，不要半途而废，不妨换一种思路，可能会出现“东方不亮西方亮”的效果。相信读者大都有数据结构和基础算法的学习经历，有熟悉语言功能和编程操作的有利条件，应该充分利用这些第一手经验加深对算法概念的理解，用实践经验促进理论学习。“纸上得来终觉浅，绝知此事要躬行。”建议读者在基本理解算法的基础上，不妨亲手做一做书中的试题，通过上机获得再认识和再提高。从长远来看，无论是参与竞赛，还是学习算法课程，或者从事编程工作，读者会发现面临的问题对象千奇百怪，数据模型千姿百态，问题求解的算法千变万化，究竟采用哪一种算法，设计哪一类数据结构，选择哪一种语句形式，确定哪一种测试手段或优化方法，都要求读者从实际出发，因地因时制宜。只有不断通过实践，才能渐入本书的程序分析所描述的那种佳境，并有所发现、发明和创造。“阵而后战，兵法之常，运用之妙，存乎一心也。”

感谢上海市复旦大学附属中学李天翼、华东师范大学第二附属中学寿国威等同学。他们在我们的指导下，花费两年多的时间读懂了算法原理，同时用简洁易识的程序代码实现了算法。所有程序通过了数据测试，其中一些程序还经过多次修改，精益求精。他们为本书的顺利修订再版付出了辛勤的劳动，做出了不可或缺的贡献。

由于时间和水平所限，书中难免存在不妥和错误之处，欢迎同仁或读者赐正。如果在阅读中发现问题，请通过书信或电子邮件告诉我们，以便及时整理成勘误表放在http://admis.fudan.edu.cn/publications.htm上，供广大读者查询更正。我们更期望读者对本书提出建设性意见，以便修订再版时改进。读者可通过以下方式与我联系。

通信地址：上海市邯郸路220号复旦大学计算机科学与工程系 吴永辉 邮编200433

电子邮箱：yhwu@fudan.edu.cn

王建德 吴永辉

2008年1月于上海

审稿人简介

陈越，浙江大学计算机学院教授，计算机学院、软件学院副院长。主讲数据结构与算法分析课程 9 年，参与编写或改编相关中英文教材 3 部。自 2000 年以来，负责浙江大学 ACM 国际大学生程序设计竞赛（ICPC）代表队的训练和组织工作，连续几年率队进入世界总决赛。自 2001 年起每年组织浙江大学程序设计竞赛、浙江省大学生程序设计竞赛等赛事，2005 年作为负责人组织了 ACM-ICPC 亚洲区杭州赛区的比赛。8 年来组织的竞赛共涉及参赛学校 300 余校次，师生 5000 余人次，受到学生广泛欢迎。

目　录

第1章 绪 论

什么是算法？为什么要对算法进行研究？相对于其他信息技术来说，算法的作用是什么?在实际生活中，算法有什么应用价值？衡量一个算法好坏的标准是什么？本章将围绕这些问题展开讨论。

1.1 算法的基本定义

简单来说，所谓算法（algorithm）就是明确的计算过程，它取一个或一组值作为输入，并生成一个或一组值作为输出。亦即，算法就是一系列的计算步骤，用来将输入数据转换成输出结果，如图1-1所示。

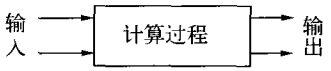

图1-1 算法图示

我们还可以将算法看作是一种工具，用来解决可以抽象出计算模型的问题。在表述该问题时，必须用严谨的语言规定所需的输入/输出关系。与之对应的算法则描述了一个特定的计算过程，用于实现这一输入/输出关系，如下所示。

输入：由n个数构成的一个序列$(a_1, a_2, \cdots, a_n)$。

输出：输出一个重排的序列$(a_1', a_2', \cdots, a_n')$，使得$a_1' \leqslant a_2' \leqslant \cdots \leqslant a_n'$。

需要指出的是，问题的表述方式是多样化的，并不一定像上面例子这么抽象呆板，例如，许多试题就采用了生动趣味的故事形式。而这里所说的语言严谨，是针对抽象计算模型而言。也就是说求解的目标是什么，给出了哪些已知信息、显式条件或隐含条件，最后应该用什么样的数据形式表达计算结果，必须描述清楚，切不能模棱两可或者产生歧义。同样，与问题对应的计算过程可以用自然语言、程序设计语言甚至硬件设计等形式来表达。不论采用哪种形式，解决问题的每一个步骤都必须准确定义，这是由于我们是和计算机打交道，稍有含糊则风马牛不相及。自然语言传递的信息，从语意上来看，可能会有不明之处，但我们处理它们时可根据上下文信息或平时习惯等来推理并准确地接受它，而计算机却不能。尤其是编程时，要用程序设计的范式语言精确定义每一个步骤，千万不要误以为自己懂了计算机也会懂。

我们衡量一个算法好坏的标准主要有两个：正确性和时效性。

(1) 算法的正确性。如果一个算法使其每一个输入实例都能在输出正确的结果后停止，则称

它是正确的。不正确的算法可能导致某些输入根本不会停止，或者停止时给出的不是预期的结果。在大学生的编程实践活动中，对算法正确性的要求是相当严格的。例如，世界上各个著名高校经常举办网上竞赛活动，由于来自世界各国的数万名参赛者的输出文件必须通过校园网统一的评测系统测试，因此参赛者的答案包括输出格式必须正确无误。当然，在编程实践活动中，有时也出现近似算法题，教师根据学生程序的运行结果与最佳解和最优效率的接近程度进行评价。这样做的目的：一是体现算法的实用性，因为有些算法虽然不是精确的，但如果其错误率能够被控制，仍不乏实用价值；二是为了让学生从推导最佳解的逻辑演绎中解放出来，通过多样化的解题途径激发其探索精神与创新意识，培养其分析解决问题的能力。当然，在大学生的编程实践活动中，求近似解这样的例外情况并不多，多数例题是求最优解的，因此，编程者的注意力还是应该集中在算法的正确性上。

(2) 算法的时效性。运行程序最重要的资源有两个：计算时间和存储空间。尽管计算机的速度很快，存储器空间很大，但计算机的速度不可能无限快，存储器空间也不可能无限大。因此，计算时间和存储空间都是有限的资源。这些有限的资源必须被有效地使用，而那些时间和空间上有效率的算法就有助于达到目的。

在编程解题中，由于试题的数据规模一般不是很大，而编程语言Free Pascal和C++提供了相对充足的用户空间，因此存储空间限制的矛盾一般不突出。而在时间效率方面，一般都有特别的要求。这是因为解决同一问题，采用不同的算法，其时间效率可能会相差很大。这种时效上的差距源自编程者知识和能力上的差距，其影响往往比硬件设备和系统软件方面的差距还要大。因此，无论今后计算机怎样更新换代，提高算法时效永远是编程者孜孜以求的目标。

1.2 算法的空间复杂度

一个算法的效率除了计算工作量外还包括占用内存空间。这里的内存空间主要指算法处理的数据所占的内存空间。除此之外，还包括算法程序所占的空间以及算法在执行过程中所需要的额外空间。其中额外空间包括算法程序执行过程中以及某种数据结构所需要的附加存储空间(例如，链结构中，除了数据本身外，还要额外存储链接信息)。算法所需要的内存空间又称算法的空间复杂度。算法需要的内存空间超出用户可用空间显然是不可行的。为了降低算法的空间复杂度，通常用到两种基本技术：压缩存储技术和原地工作。

1.2.1 压缩存储技术

在许多情况下，算法处理的数据量是很大的。为了尽量减少数据所占的存储空间，往往需要对数据进行压缩。例如，采用相邻矩阵$A=(a_{ij})$存储具有n个顶点的图$G=(V,E)$。

$$a_{ij}=\begin{cases}1 & (i,j)\in E\\ 0 & (i,j)\notin E\end{cases} \quad (1\leqslant i,j\leqslant n)$$

无向图的相邻矩阵是对称的。在一些应用中，通常将其相邻矩阵对角线以上的部分均设为0。

$$a_{ij}=\begin{cases}0\text{ 或 }1 & (j\leqslant i)\\ 0 & (j>i)\end{cases}$$

因此可以只存储左下三角的非零元素，这样既节省了存储空间，又不会丢失任何数据。在n阶左下三角矩阵A中，左下三角的非零元素最多为$\left\lfloor\frac{n(n+1)}{2}\right\rfloor$个。假设用一维数组$b$：

$$b=\left[1..\left\lfloor\frac{n(n+1)}{2}\right\rfloor\right]$$

以行为主存储左下三角中的元素，且左下三角中的元素a_{ij}（$i\geqslant j$）在一维数组b中为第k个元素，则容易推出k与i、j的关系为

$$k=\frac{i(i-1)}{2}+j\qquad(j\leqslant i)$$

显然这种存储结构节省了大约一半的空间，而且存取原理也十分简单。如果要访问图中的某条边，若该边属于相邻矩阵A中的左下三角元素，则访问一维数组b中与之对应的元素；若该边属于相邻矩阵A中的右上三角元素，则不再访问。通过以下对应关系实现对相邻矩阵A的访问。

$$a_{ij}=\begin{cases}b\left[\frac{i(i-1)}{2}+j\right] & (j\leqslant i)\\ 0 & (j>i)\end{cases}$$，其中，$1\leqslant i,\ j\leqslant n$

当然，这种压缩存储技术既有节省内存的“利”，也有增加计算时间的“弊”。因为每访问图中的一条边，需要通过上述公式计算该边对应的一维数组b的下标。

1.2.2 原地工作

前已述及，一个算法所占用的存储空间还包括了算法在执行过程中所需要的额外空间。如果算法执行过程中所需要的空间相对于问题规模来说是常量，即该额外空间不随问题规模的增大而增大时，则称此算法为原地工作的。例如，在相邻矩阵A的基础上计算任意顶点对之间是否有路（传递闭包），使得

$$a_{ij}=\begin{cases}1 & \text{顶点}i\text{与顶点}j\text{之间有路}\\ 0 & \text{顶点}i\text{与顶点}j\text{之间无路}\end{cases}$$

就属于一种原地工作。

总之，为了降低算法的空间复杂度，应在兼顾时间效率的同时，尽量采用压缩存储技术，并尽量使算法原地工作。

1.3 算法的时间复杂度

所谓算法的时间复杂度是指执行算法的计算工作量。一个算法的有穷性实际上包含了合理的执行时间的含义，这就是算法工作量分析的问题。实际上，在解决一个问题时可能存在几种方法，

而在这几种方法中，所需要的计算工作量又可能是不一样的，而我们总是希望计算工作量越小越好，这就有一个选择的问题。

在分析算法工作量时，首先遇到的一个问题是如何衡量算法的工作量。一种最简单且方便的方法是将执行时间作为算法工作量的度量。但由于算法的执行时间往往与所使用的计算机有关，而我们又不可能针对某台具体的计算机来分析算法工作量，因此这种度量方法是不可取的。另一种方法是用算法程序中所执行的指令条数或语句条数来度量。但很明显，这种方法又在很大程度上取决于所使用的程序设计语言以及程序设计者的水平和风格，因此也是不可取的。看来，从运算的执行次数来度量似乎合理一些，然而执行次数也因基本运算、输入规模和输入情况有很大差别。现分述如下。

1.3.1 基本运算

计算机里可实施的运算有算术运算和逻辑运算。一次乘法的运算时间是一次加法运算时间的40～60倍。所以撇开运算的性质只谈次数无法比较算法的优劣，只好在相同性质的运算情况下讨论问题的算法，也就是确定该问题的基本运算。

基本运算是一个算法中运算量最大的一种运算，是该算法运算的主要特征，如表1-1所示。

表1-1 基本运算举例

问 题	基本运算
在一维数组中找x	x和数组元素比较
实数矩阵相乘	实数乘法
排序	表中元素相互比较
图的运算	访问顶点

事实上每种算法除了基本运算外还有其他运算。例如，实数矩阵相乘运算中含有加法，虽然计算量和乘法比起来不占主导地位，但也有一定工作量。对于不能略去的非基本运算，算法分析中乘上加权因子（大于1的系数），然后将其与基本运算次数的和作为算法工作量。

1.3.2 输入规模

确定了基本运算，讨论就有了前提。但运算量还和输入量有关，同是图的计算，5个顶点、10条边的图与20个顶点、40条边的图相比，运算量也许要差出一个数量级。所以对每种基本运算还需要一种衡量问题输入规模的度量。以表1-1为例，列出表1-2。

表1-2 输入规模举例

问 题	输入规模
在一维数组中找x	数组元素个数n
实数矩阵相乘	矩阵的阶nm
排序	表中元素个数n
图的运算	顶点数n，边数m

1.3.3 输入情况

确定了输入规模还不够，细讨论起来还要考虑具体输入情况。例如，“在一维数组中找出等于x的数”这个题目。如果输入的一维数组中第1个数就等于x和第n个数才是x，其查找次数比是1:n。所以同一个算法，可以从不同的输入情况讨论它的时间复杂度。这样看来情况太多，讨论起来就很复杂了。于是，对每一种算法均从两种情况来讨论它的时间复杂度。

1. 平均情况

每种算法都可能有不同的输入。但从概率统计的观点来看，它有一个平均的输入情况。平均情况下算法工作量可用下面的公式表示。其中，$A(n)$为输入规模为n的平均算法工作量，用基本运算次数计算；$D(n)$为可能的输入集合；$P(i)$为输入集合中第i种输入可能发生的概率；$T(i)$为输入集合中第i种输入情况下所需计算工作量。

$$A(n)=\sum_{i\in D(n)} P(i)T(i)$$

这样求加权和比较精确，因此对详细分析算法是有意义的。但是$P(i)$往往难以估计，况且抽样统计每种输入情况极其麻烦，因此人们常用最坏情况度量算法工作量。

2. 最坏情况

最坏情况估算起来容易得多，它只在所有可能输入的计算工作量中取最大的一种来估算，即

$$O(n)=\max_{i\in D(n)}\{T(i)\}$$

显然这样估出的算法的时间复杂度是保守的，它给出时间复杂度的上界，但在实际应用中很方便，一是估算容易，二是可估出上机需要的最长时间。所以大多数算法分析给出的都是最坏情况。下面是这两种算法分析的实例。

例1.1 顺序查找

从n（$1\leqslant n\leqslant 1\,000$）个元素的整数数组$l$中查找元素$x$，基本运算为数组元素和$x$比较。

输入：$l[1..n]$和被查元素x。

输出：若x不在数组l中，则返回0；否则返回l中第1个使得$l[j]=x$的下标j。

思路点拨

设数组l的类型定义如下：

```
Type
    ltype=array[1..1000] of integer;
```

算法程序如下：

```
function search(l:ltype;x:integer):integer;
Var j:integer;
{
    j←1;
```

```
    while (j<=n)and(l[j]<>x)do j←j+1;
    if j>n then search←0
    else search←j;
}
```

(1) 平均情况分析

x可能出现在数组l的任何位置上，因此可能的输入有$\{l_1,\cdots, l_n, l_{n+1}\}$。其中$l_{n+1}$为$x$不在$l$中的第$n+1$种情况。假定第$i$次输入$x$放在第$i$个位置上，$q$为$x$在$l$中出现的概率。若$x$在$l$中，则比较次数$T(l_i)=i$，按均匀分布考虑，出现在第$i$个位置的概率为$P(l_i)=\frac{q}{n}$（$1\leqslant i\leqslant n$）；若$x$不在$l$中，则$T(l_{n+1})=n$，$P(l_{n+1})=1-q$。于是有

$$A(n)=\sum_{i=1}^{n+1}p(l_i)T(l_i)=\sum_{i=1}^{n}\frac{q}{n}i+(1-q)n=\frac{q(n+1)}{2}+(1-q)n=\frac{q}{2}+n-\frac{qn}{2}$$

如果x肯定在数组l中（$q=1$），则$A(n)=\frac{n+1}{2}$，即说明在n个元素的数组中查找x，平均要查找半个数组；如果x有一半机会不在数组l中（$q=\frac{1}{2}$），则$A(n)=\frac{1+3n}{4}\approx\frac{3n}{4}$（$n$足够大时），即说明平均要查3/4的数组长度。

(2) 最坏情况分析

最坏情况下，我们查找整个数组。

$$O(n)=\max\{T(l_i)\mid 1\leqslant i\leqslant n+1\}=n$$

综上所述，一个算法的工作量通常取决于问题的规模和具体输入的性质。但有些算法的工作量与输入无关。当一个输入规模为n的问题的算法工作量与具体输入无关时，它对于所有可能的输入有：$A(n)=O(n)$，同时也等于算法所做的基本运算次数。

1.3.4 时间复杂度的阶

如果一个问题有多种算法，应如何选择算法？例如，下述两个算法中谁的效率高呢？

$$O_1(n)=n^2 \qquad O_2(n)=25n$$

很显然，当输入规模$n<25$时，第一种算法比第二种算法快；当$n\geqslant 25$时正好相反。算法分析考虑一般情况$n\to\infty$。当$n>n_0$时，如果存在正的常量C（即考虑除基本运算外所必需的附加操作），使得$O_1(n)\leqslant O_2(n)C$，则称O_1比O_2低阶或它们同阶；如果同样也有O_2比O_1低阶或它们同阶，则称O_1与O_2是同阶的。例如，$O_1(n)=\frac{n^2}{2}$和$O_2(n)=370n^2$是同阶的。由此看出如果两种最坏情况分析之间相差常量因子，则它们是同阶的。如果一个问题有多种算法，我们首先寻求低阶的算法，然后再来解决那些可以减少工作量但不影响阶的细节问题。

也许有人会提出，由于现代计算机的进步，计算机速度的增长将会削弱有效算法的重要性。然而情况恰恰相反。为了说明问题，列举5个算法A_1、A_2、A_3、A_4和A_5。这些算法按阶的递增顺

序排列。假定一个单位时间（频数）等于一毫秒，即一毫秒内可以处理一个问题个数为1的输入。表1-3列出了这5个算法在一秒、一分钟和一小时内可能解决的问题规模以及计算机速度增加10倍后的情形。

表1-3 算法时效与计算机增速之间的关系

算法	时间复杂度	问题的最大规模			速度增加10倍后可解决问题的最大规模
		一秒钟	一分钟	一小时	
A_1	n	1 000	6×10^4	3.6×10^6	10倍
A_2	$n\,\mathrm{lb}\,n$	140	4 893	2.0×10^5	接近10倍
A_3	n^2	31	244	1 897	3.16倍
A_4	n^3	10	39	153	2.15倍
A_5	2^n	9	15	21	增加3.3个问题

由表1-3可以看出，算法A_1在一秒钟里可以处理一个规模为1 000的输入，而算法A_5在一秒钟里仅能处理一个规模至多为9的输入。换上了速度快10倍的计算机，算法A_5解决问题输入的个数增加了3，而算法A_1可以解决问题规模大10倍的问题。暂不考虑速度的增长，从表中一分钟一栏可以看出，用算法A_3替换A_4（降低一个阶数），可以解决问题规模大6倍的问题。用算法A_2来替换A_4（降低两个阶数），可以解决问题规模大125倍的问题，而这些效果比计算机速度增加10倍更为显著，如果以一小时作为比较基础，则效果就更明显了。即便是计算机的速度增加100倍，低价算法依然显示其优越性。我们不妨再来看一个例子。

有两个排序算法，若以1次比较看作一个时间单位，对随机产生的n个数据项进行排序，插入排序算法的时间约为n^2，而快速排序算法所需的时间大约是$n\,\mathrm{lb}\,n$，显然$n>\mathrm{lb}\,n$，因此在同一台计算机上，快速排序比插入排序运行得快。现在有两台计算机：计算机A的速度快，每秒能执行10^9条指令，运行插入排序；计算机B的速度慢，每秒只能执行10^7条指令，运行快速排序。在计算能力方面，计算机A要比计算机B快100倍。为了使这一差距更为明显，假设让世界上最能干的程序员采用机器语言为计算机A编写插入排序算法的代码，所得到的代码需要n^2条指令；另一方面，让一位平均熟练水平的程序员，采用某种具有低效编译器的高级语言为计算机B编写快速排序算法的代码，所得到的代码有$n\,\mathrm{lb}\,n$条指令。为了排序10^6个数，计算机A花的时间为

$$\frac{(10^6)^2\text{条指令}}{10^9\text{条指令/s}}=1\,000\ \mathrm{s}$$

计算机B花的时间为

$$\frac{(10^6)\mathrm{lb}\,10^6\text{条指令}}{10^7\text{条指令/s}}\approx 2\ \mathrm{s}$$

计算机B由于采用了一种运行时间增长得更为缓慢的算法，尽管它用的是效率较低的编译器，运行速度却比计算机A快了500倍！需要排序的数据个数愈多，快速排序的优势就愈加明显了。

由此可见，时间复杂度的阶是一个重要的性能标准，寻求低价算法比提高计算机的运行速度更为必要。